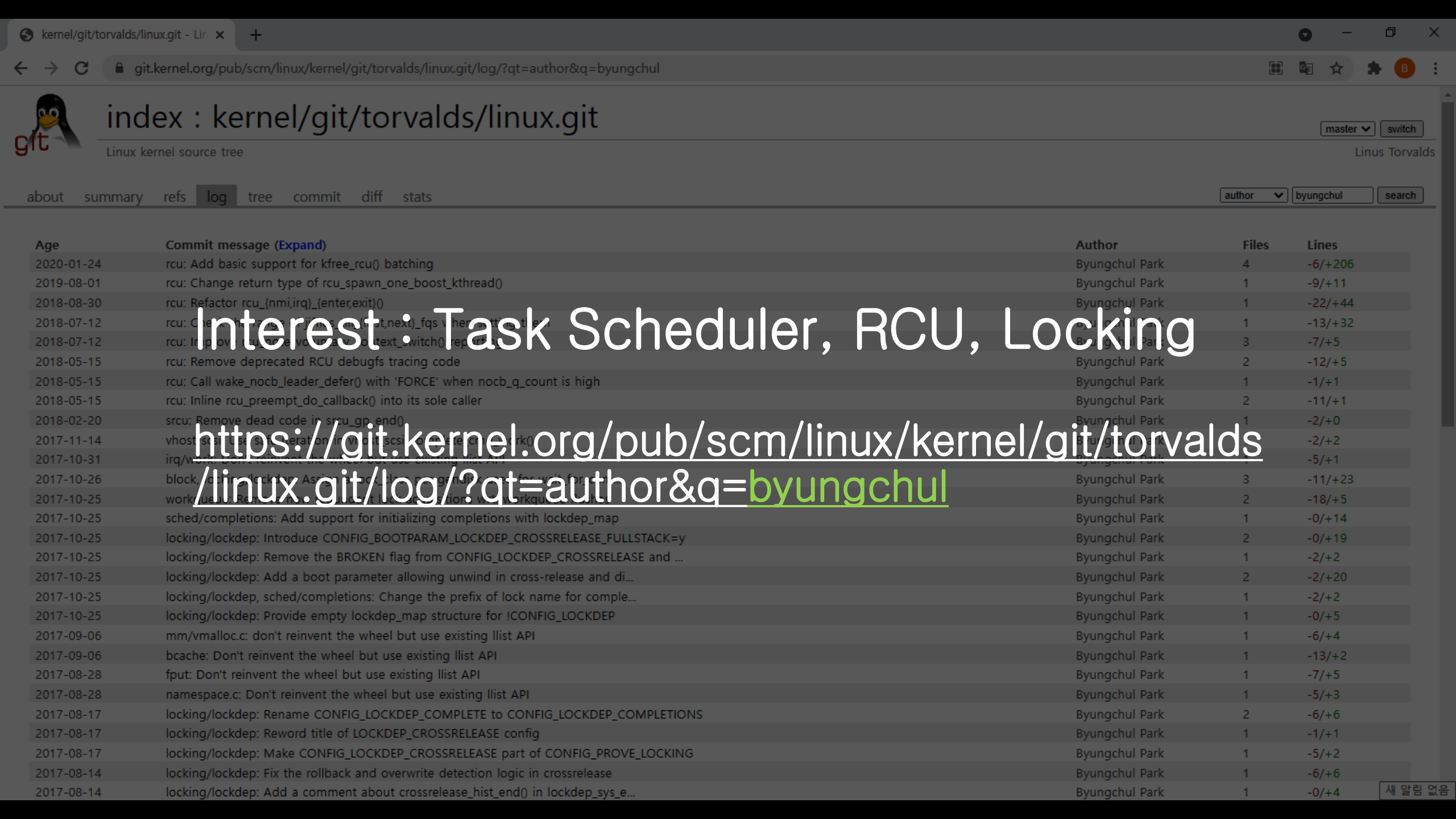
DEPT Dependency Tracker

What Lockdep does and doesn’t

2022 . 9 . 13

Byungchul Park || LG Electronics

max.byungchul.park@gmail.com



Interest : Task Scheduler, RCU, Locking

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/log/?qt=author&q=byungchul

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

// Holds A.

spin_lock A

Lock / unlock balance

...

Lock and unlock must be used in pairs.

// Terminates the task without releasing A.

exit

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds A.
spin_lock A
...

// Terminates the task without releasing A.
exit
```

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds A.
spin_lock A
...

// Terminates the task without releasing A.
exit
```

REPORT!

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds A.
spin_lock A
...

// Returns to user without releasing A.
return to user
```

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds A.
spin_lock A
...

// Returns to user without releasing A.
return to user
```

REPORT!

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds spin lock A.
spin_lock A

...

// Holds mutex lock B.
mutex_lock B

...

mutex_unlock B

spin_unlock A
```

Relation between lock types

mutex_lock shouldn’t be acquired in a spin_lock critical section.

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds spin lock A.
spin_lock A
...
// Holds mutex lock B.
mutex_lock B
...
mutex_unlock B
spin_unlock A
```

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds spin lock A.
spin_lock A
...
// Holds mutex lock B.
mutex_lock B
...
mutex_unlock B
spin_unlock A
```

REPORT!

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds raw spin lock A.
raw_spin_lock A

...

// Holds spin lock B.
spin_lock B

...

spin_unlock B
raw_spin_unlock A
```

Relation between lock types

spin_lock shouldn’t be acquired in a raw_spin_lock critical section.

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds raw spin lock A.
raw_spin_lock A
...
// Holds spin lock B.
spin_lock B
...
spin_unlock B
raw_spin_unlock A
```

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds raw spin lock A.
raw_spin_lock A
...
// Holds spin lock B
spin_lock B
...
spin_unlock B
raw_spin_unlock A
```

REPORT!

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

// Holds A.
spin_lock A

Nest lock usage

// Holds B with nest lock A.
spin_lock_nest_lock B, A

Nest lock API should be called with the nest lock held.

...
spin_unlock B
spin_unlock A

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds A.
spin_lock A
...
// Holds B with nest lock A.
spin_lock_nest B, n=A
...
spin_unlock B
spin_unlock A
```

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds B with nest lock A.
spin_lock_nest B, n=A
...
spin_unlock B
```

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds B with nest_lock A
spin_lock_nest B, n=A
...
spin_unlock B
```

REPORT!

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

RCU read forbidden context

// Specifies a RCU read section.

rcu_read_lock

Usage from illegal states

RCU read forbidden context e.g. idle shouldn’t have RCU read sections.

rcu_read_unlock

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

RCU read forbidden context

```
// Specifies a RCU read section.
rcu_read_lock
...
rcu_read_unlock
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

RCU read forbidden context

// Specifies a RCU read section.

rcu_read_lock

...

rcu_read_unlock

REPORT!

Proper use of the APIs

rcu_dereference_protected is supposed to be used with updater lock held.

rcu_dereference is supposed to be used in RCU read section.

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Specifies a RCU read section.
rcu_read_lock
...
// Calls without necessary lock held.
rcu_dereference_protected A
...
rcu_read_unlock
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
// Specifies a RCU read section.
rcu_read_lock
...
// Calls without necessary lock held.
rcu_dereference_protected A
...
rcu_read_unlock
```

REPORT!

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

// Specifies a RCU read section.

rcu_read_lock

Delimitation for read section

rcu_read_unlock

RCU read APIs should be used in a section delimited by rcu_read_lock / rcu_read_unlock.

// Traverse elements out of read section.

list_for_each_entry_rcu

...

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
// Specifies a RCU read section.
rcu_read_lock
...
rcu_read_unlock
...
// Traverses RCU list out of read section.
list_for_each_entry_rcu
...
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
// Specifies a RCU read section.
rcu_read_lock
...
rcu_read_unlock
...
// Traverses RCU list out of read section.
list_for_each_entry_rcu
...
```

REPORT!

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds A.
spin_lock A
// Holds B while holding A.
spin_lock B
spin_unlock B
spin_unlock A
```

```
// Holds B.
spin_lock B
...
// Holds A while holding B.
spin_lock A
spin_unlock A
spin_unlock B
```

Lock acquisition order

All the lock acquisition order should be kept across the kernel code.

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds A.
spin_lock A
...
// Holds B while holding A.
spin_lock B
...
spin_unlock B
spin_unlock A
```

```
// Holds B.
spin_lock B
...
// Holds A while holding B.
spin_lock A
...
spin_unlock A
spin_unlock B
```

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds A.
spin_lock A
…
// Holds B while holding A
spin_lock B
…
spin_unlock B
spin_unlock A
```

```
// Holds B.
spin_lock B
…
// Holds A while holding B.
spin_lock A
…
spin_unlock A
spin_unlock B
```

REPORT!

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

// Holds A.

spin_lock A

...

// Holds A while holding A.

spin_lock A

Lock recursive acquisition

The same lock shouldn’t be acquired twice in a single context.

spin_unlock A

...

spin_unlock A

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
// Holds A.
spin_lock A
...
// Holds A while holding A.
spin_lock A
...
spin_unlock A
...
spin_unlock A
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
// Holds A.
spin_lock A
...
// Holds A while holding A.
spin_lock A
...
spin_unlock A
...
spin_unlock A
```

REPORT!

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

interrupt context

spin_lock_irq A

...

Irq safe usage

Irq safe APIs should be used if the lock is used in either process or interrupt context.

// Holds A with irq disabled.

spin_lock_irq A

...

spin_unlock_irq A

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

interrupt context

```
spin_lock_irq A
…
spin_unlock_irq A
```

process context

```
// Holds A with irq disabled.
spin_lock_irq A
…
spin_unlock_irq A
```

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

interrupt context

```
spin_lock A
...
spin_unlock A
```

process context

```
// Holds A with irq enabled.
spin_lock A
...
spin_unlock A
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

process context

```
// Holds A with irq enabled.
spin_lock A
...
        interrupt context
        spin_lock A
        ...
        spin_unlock A
spin_unlock A
```

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

process context

// Holds A with irq enabled.

spin_lock A

…

interrupt context

spin_lock A

…

spin_unlock A

spin_unlock A

REPORT!

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

Lockdep is a quite good tool for checking these.

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
mutex_lock A
...
// Waits for event B.
wait_for_event B
...
// Releases A.
mutex_unlock A
```

```
// Waits for A to be released.
mutex_lock A
...
mutex_unlock A
...
// Wakes up wait_for_event B.
event B
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
mutex_lock A
…
// Waits for event B.
wait_for_event B
…
// Releases A.
mutex_unlock A
```

```
// Waits for A to be released.
mutex_lock A
…
mutex_unlock A
…
// Wakes up wait_for_event B.
event B
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
mutex_lock A
…
// Waits for event B.
wait_for_event B
…
// Releases A.
mutex_unlock A
```

```
// Waits for A to be released.
mutex_lock A
…
mutex_unlock A
…
// Wakes up wait_for_event B.
event B
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
mutex_lock A
…


…
// Releases A.
mutex_unlock A
```

```
// Waits for A to be released.
mutex_lock A
…
mutex_unlock A
…
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
mutex_lock A
...
                    // Waits for A to be released.
                    mutex_lock A
                    ...
                    mutex_unlock A
                    ...
...
// Releases A.
mutex_unlock A
```

REPORT???

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

What makes the case missed?

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

How does it track dependency?

Define Dependency

Lock A
Lock B (while holding A.)

= A depends on B. (A → B)

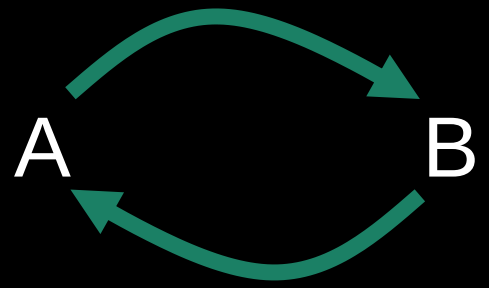
Example

```
// Holds A.
spin_lock A
…
// Holds B while holding A.
spin_lock B
…
spin_unlock B
spin_unlock A
```

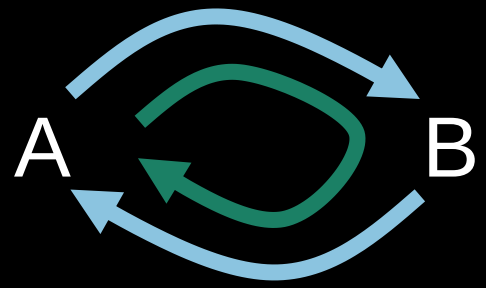
Build Dependency Graph

Example

```
// Holds B.
spin_lock B
…
// Holds A while holding B.
spin_lock A
…
spin_unlock A
spin_unlock B
```

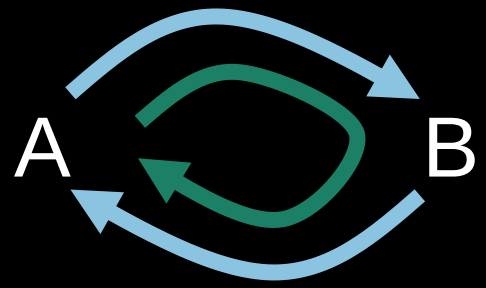


Build Dependency Graph

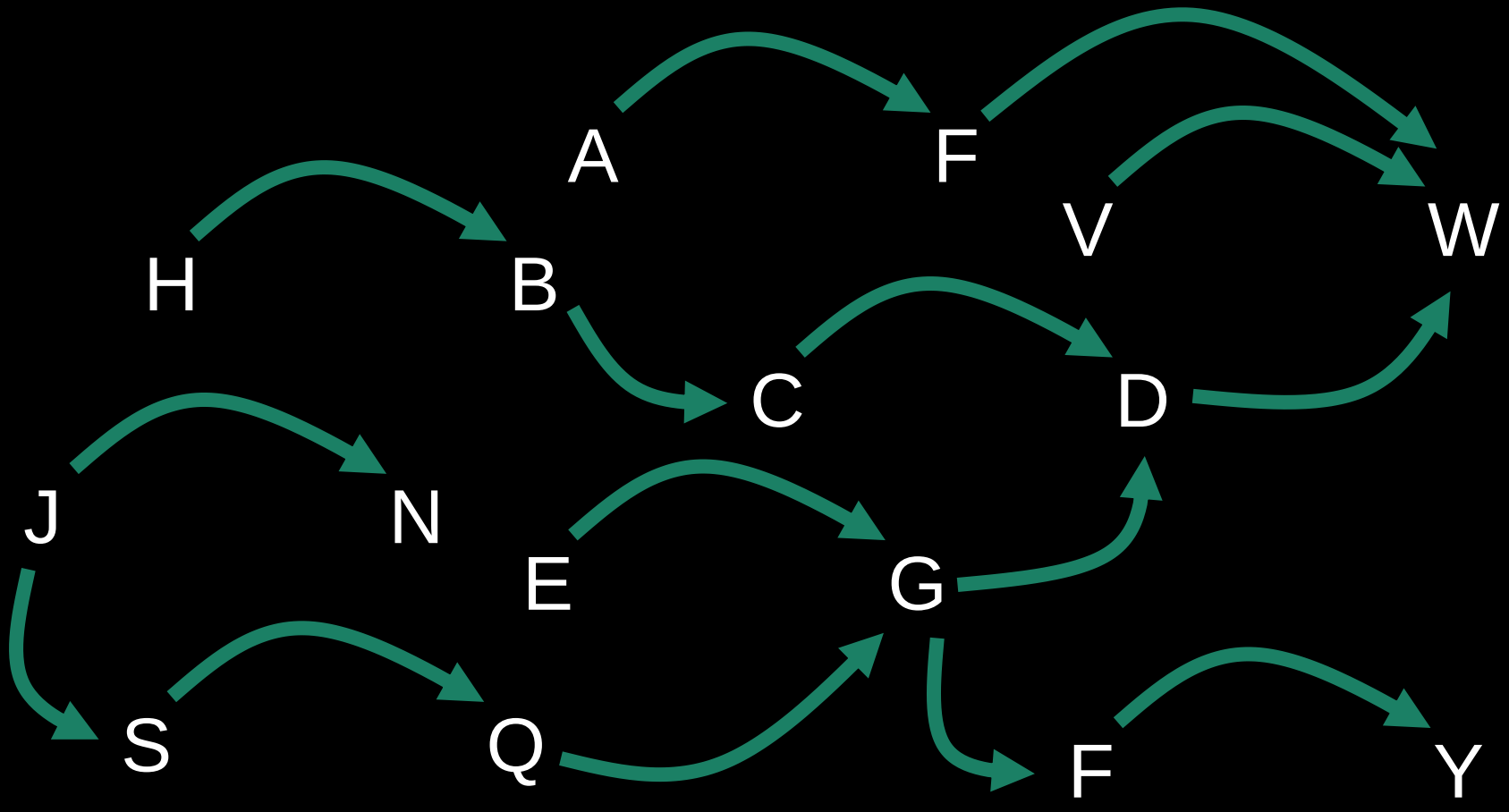


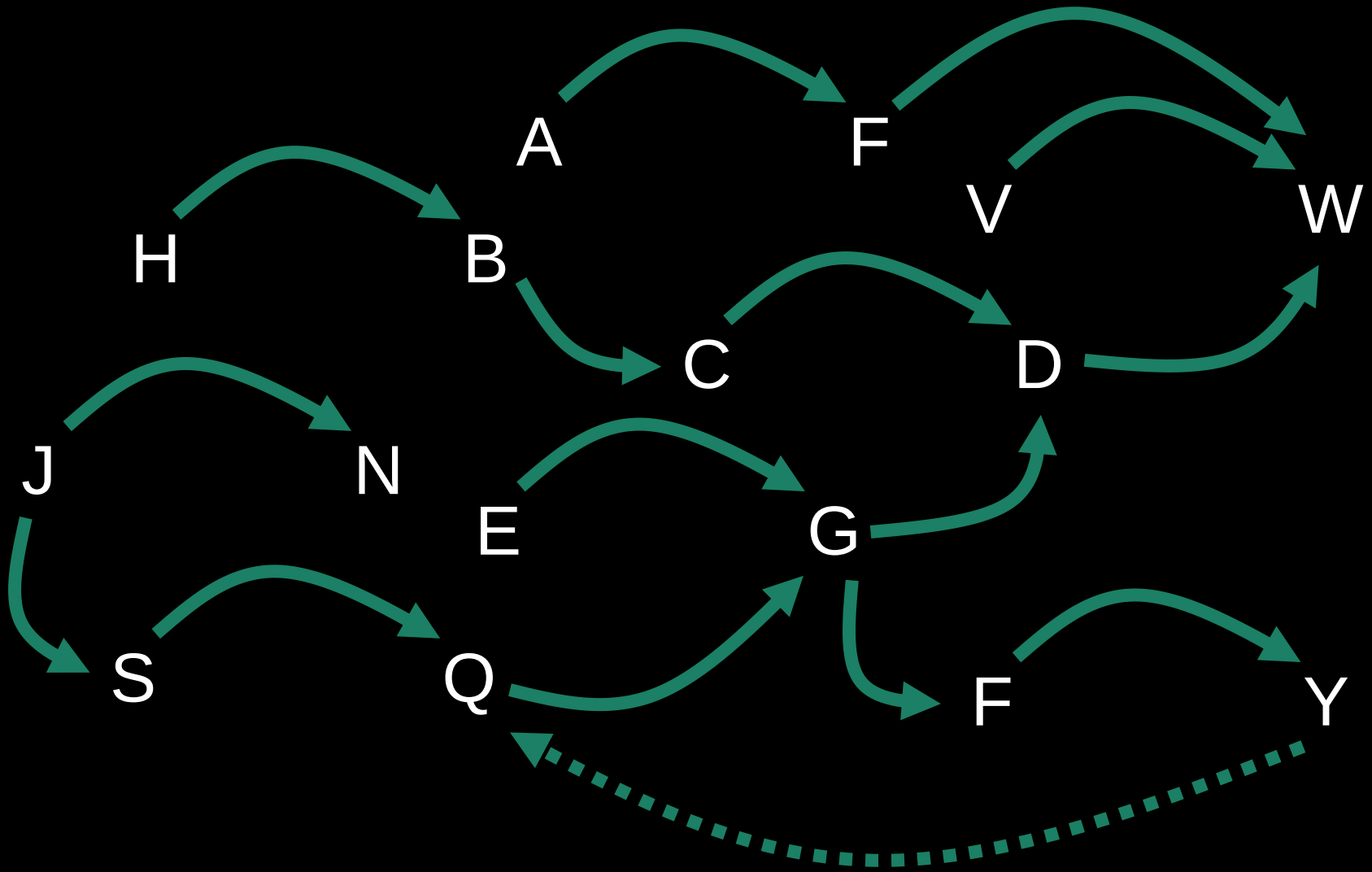
Circular Dependency

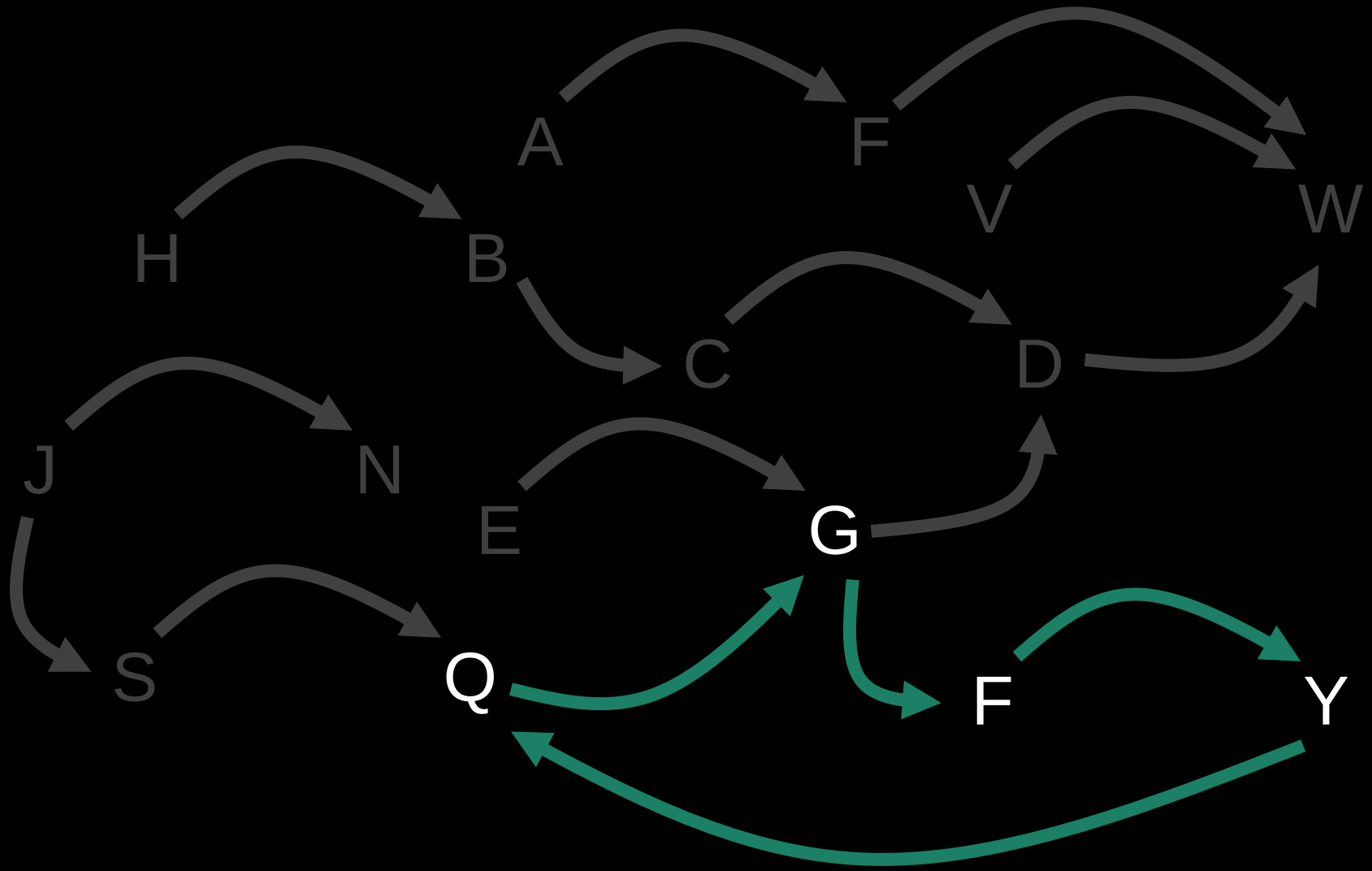
REPORT!

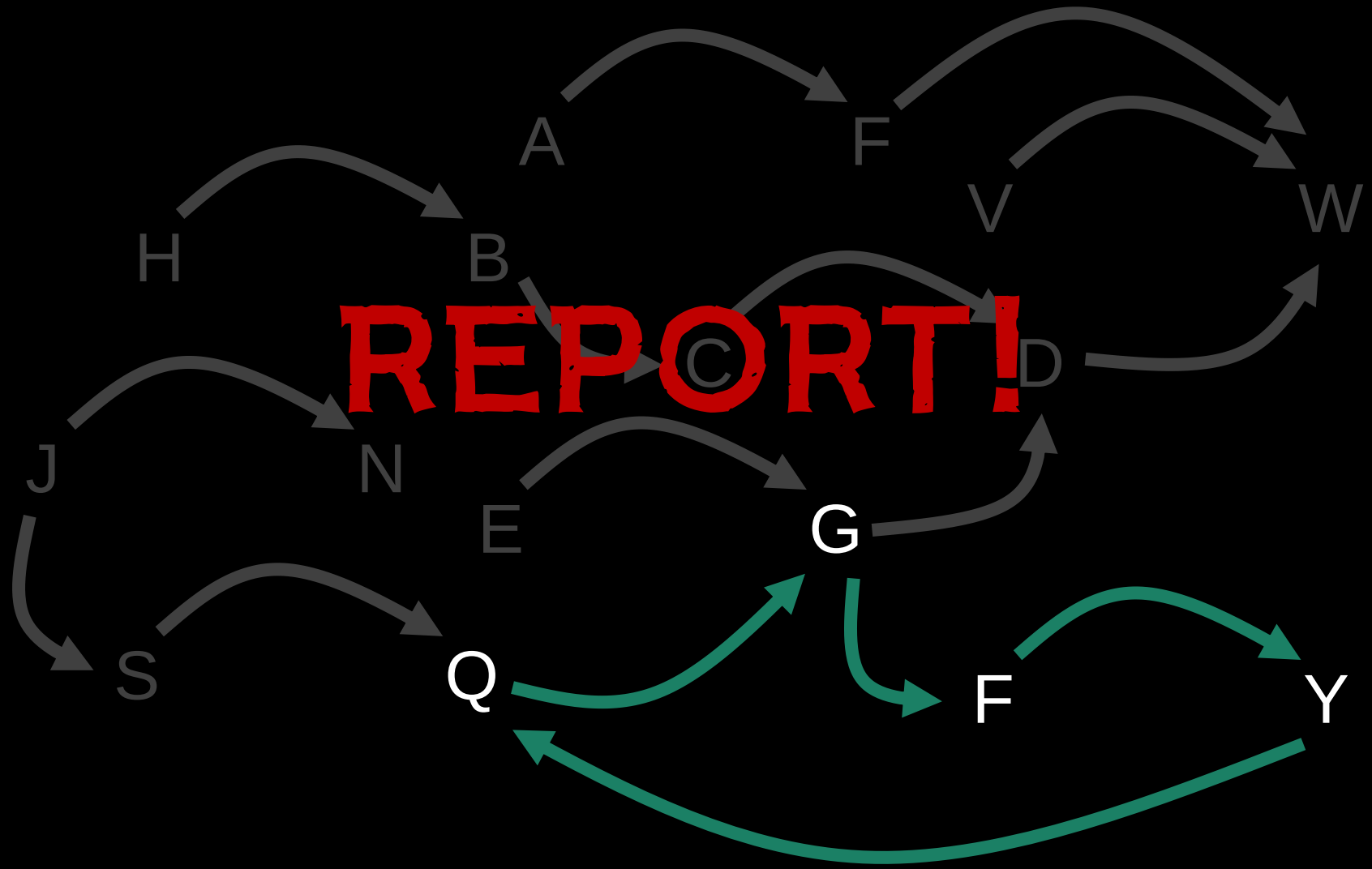


Circular Dependency


A
F
H
B
V
W
C
D
J
N
E
G
S
Q
F
Y


A
F
V
W
H
B
C
D
J
N
E
G
S
Q
F
Y


A
F
H
B
V
W
C
D
J
N
E
G
S
Q
F
Y


A
F
H
B
V
W
REPORT!
C
D
J
N
E
G
S
Q
F
Y

Is a deadlock caused by
incorrect lock acquisition order?

Is a deadlock caused by
incorrect lock acquisition order?

Partially yes.

A deadlock is caused by

Waiters that cannot be woken up.

A deadlock is caused by

Events that are not reachable.

Deadlock detection tool should focus on waits and events themselves.

Define Dependency

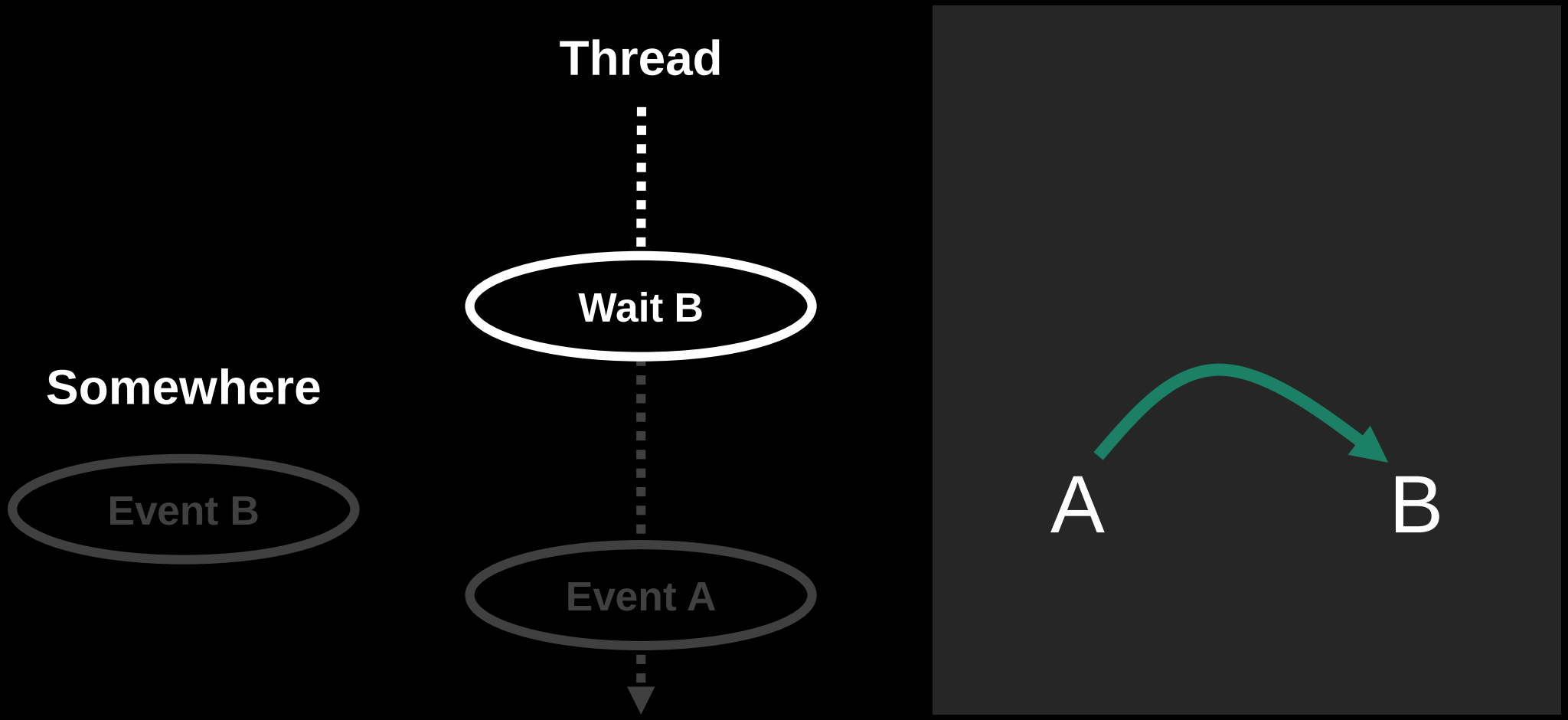
Lock A
Lock B (while holding A.)

= A depends on B. (A → B)

Redefine Dependency

Event A occurrence depends on
Event B occurrence.

= A depends on B. (A → B)



Event A occurrence depends on Event B occurrence.

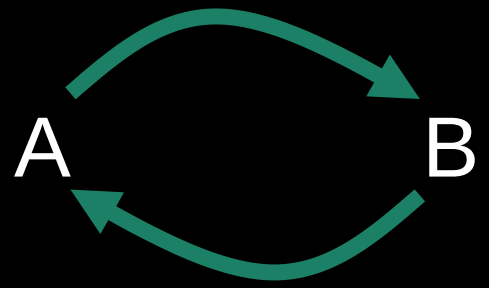
Example

```
mutex_lock A
…
// Waits for event B.
wait_for_event B
…
// Depends on event B.
mutex_unlock A
```

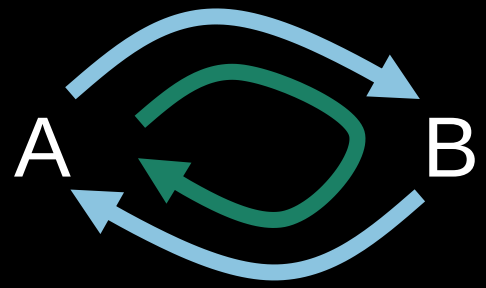
Build Dependency Graph

Example

```
// Waits for A to be released.
mutex_lock A
…
mutex_unlock A
…
// Depends on releasing A.
event B
```

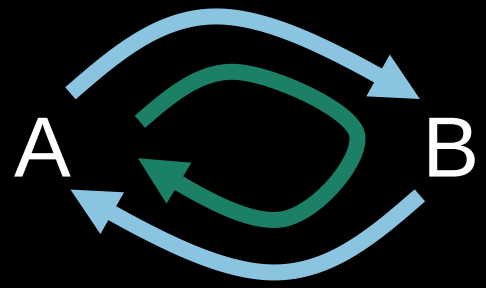


Build Dependency Graph



Circular Dependency

REPORT!



Circular Dependency

REWIND ...

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
// Holds A.
spin_lock A
...
// Holds B while holding A.
spin_lock B
...
spin_unlock B
spin_unlock A
```

```
// Holds B.
spin_lock B
...
// Holds A while holding B.
spin_lock A
...
spin_unlock A
spin_unlock B
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
// Holds A.
spin_lock A
...
// Holds B while holding A.
spin_lock B
...
spin_unlock B
spin_unlock A
```

waiter B

event A

```
// Holds B.
spin_lock B
...
// Holds A while holding B.
spin_lock A
...
spin_unlock A
spin_unlock B
```

waiter A

event B

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
// Holds A.
spin_lock A
...
// Holds A while holding A.
spin_lock A
...
spin_unlock A
...
spin_unlock A
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
// Holds A.
spin_lock A
...
// Holds A while holding A.
spin_lock A
...
spin_unlock A
...
spin_unlock A
```

waiter A

event A

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

process context

```
// Holds A with irq enabled.
spin_lock A
...
        interrupt context
        spin_lock A
        ...
        spin_unlock A
spin_unlock A
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

process context

// Holds A with irq enabled.

spin_lock A

...

interrupt context

spin_lock A

waiter A

...

spin_unlock A

spin_unlock A

event A

- Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage
- RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section
- Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Irq safe usage

```
mutex_lock A
...
// Waits for event B.
wait_for_event B
...
// Releases A.
mutex_unlock A
```

```
// Waits for A to be released.
mutex_lock A
...
mutex_unlock A
...
// Wakes up wait_for_event B.
event B
```

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```
mutex_lock A
...
// Waits for event B.
wait_for_event B
...
// Releases A.
mutex_unlock A
```

waiter B

event A

```
// Waits for A to be released.
mutex_lock A
...
mutex_unlock A
...
// Wakes up wait_for_event B.
event B
```

waiter A

event B

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

Lockdep is a great tool
for checking these two.

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

Lockdep is not the best tool for this purpose.

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

So I developed a new tool named DEPT to replace the 3rd part to deal with dependency in the best way.

Lockdep is not the best tool for this purpose.

Lock usage correctness

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

So I developed a new tool named DEPT to replace the 3rd part to deal with dependency in the best way.

DEPT is the best tool for this purpose.

DEPT detects situations where waiters cannot be woken up.

DEPT detects situations where events are not reachable.

Pros

Works with all types of wait.

Supports multiple reports efficiently.

Provides easy APIs to annotate wait / event.

Cons

Just started so still has false positives.

LKML Discussion

https://lkml.org/lkml/2022/5/4/218

Github Repository

https://github.com/lgebyungchulpark/linux-dept

WANTED